Comunicação entre Objetos Distribuídos Replicados Tolerantes a Falhas

Marcelo Hanel
Graduando em Ciência da Computação - Pesquisador do Curso de Ciência da Computação Instituto de Ciências Exatas e Geociências – ICEG Universidade de Passo Fundo – UPF Km 171 BR285 Bairro São José Caixa Postal 611 CEP 99001-970 Passo Fundo – RS Fone (054) 3168100 da Universidade de Passo Fundo – UPF. Rio Grande do Sul. Brasil. e-mail: 4570@lci.upf.tche.br

Maykel Três
Graduando em Ciência da Computação - Pesquisador do Curso de Ciência da Computação Instituto de Ciências Exatas e Geociências – ICEG Universidade de Passo Fundo – UPF Km 171 BR285 Bairro São José Caixa Postal 611 CEP 99001-970 Passo Fundo – RS Fone (054) 3168100 da Universidade de Passo Fundo – UPF. Rio Grande do Sul. Brasil. e-mail: 19434@lci.upf.tche.br

Emerson Rogério de Oliveira Junior
Doutorando da Universidade Federal do Rio Grande do Sul - UFRGS e Professor da Universidade de Passo Fundo – UPF. Áreas de Pesquisa: Arquitetura de Computadores, Processamento Paralelo e Distribuído, Tolerância a Falhas em Sistemas Distribuídos. e-mail: emerson@inf.ufrgs.br

RESUMO

Este trabalho apresenta uma forma de transformar aplicações orientadas a objetos tolerantes a falhas de *crash*, podendo ser utilizado em um ambiente distribuído baseado em objetos que se comunicam entre si, a fim de executarem uma ação. Para garantir a disponibilidade de uma aplicação distribuída, os objetos distribuídos devem ser replicados. Desta forma, a ocorrência de falha de *crash* no processador que está executando um objeto distribuído não fará com que toda a aplicação deixe de ser executada. Para validar a proposta, são apresentadas duas implementações de uma mesma aplicação tolerante a falhas. Estas implementações foram realizadas com PVM e com Java-Sockets e são apresentados os resultados obtidos.

Palavras-Chave: Sistemas distribuídos, objetos distribuídos, tolerância a falhas em sistemas distribuídos, replicação, primário-backup, PVM, Java-Sockets.

ABSTRACT

This work presents a study that shows a way to transform crash fault tolerant object-oriented applications, that can be used in object-based distributed environments, whose objects communicates each other, to execute a determined action. To guarantee distributed application availability, the distributed objects must be replicated. This is necessary because crash faults occurrency on the processor where distributed object is processing may cause interruption. Two implementations are presented to validate this proposal. These implementations were executed using PVM and Java-Sockets. Finally, the obtained results are presented.

Key-Words: Distributed systems, distributed objects, fault tolerance in distributed systems, replication, primary-backup, PVM, Java-Sockets.

1 Introdução

Atualmente, os sistemas distribuídos têm sido amplamente utilizados nos sistemas de computação em geral. Este fato deve-se, principalmente, à grande difusão do uso de redes de computadores. O objetivo é prover um aumento considerável no poder de processamento.

É adotada, neste trabalho, a mesma definição de sistema distribuído apresentada por [BAL89], considerando um conjunto de processadores autônomos, sem compartilhamento de memória, cooperando entre si através de uma rede de comunicação. Esta definição abrange desde arquiteturas multiprocessadoras até redes de computadores dispersas geograficamente (WANs).

Com o aumento da utilização dos computadores, apareceu a necessidade de se ter segurança na execução de sistemas de computação. Neste contexto surge a área de tolerância a falhas.

Técnicas de tolerância a falhas são utilizadas em sistemas distribuídos visando detectar a ocorrência de erros produzidos por falhas e recuperá-los, para continuar uma operação e retornar à computação normal [JALOTE 94]. Uma das técnicas é conhecida como replicação.

A replicação permite que objetos possam ser duplicados fazendo com que, na ocorrência de falha em um objeto, uma cópia do objeto possa assumir as funções do objeto inicial.

Este artigo apresenta uma maneira de transformar aplicações distribuídas tolerantes a falhas de *crash*. Para tanto será utilizada a replicação dos objetos através da técnica do primário-backup. Para validar esta proposta foi implementada uma aplicação baseada em objetos distribuídos. Esta aplicação foi executada em dois ambientes diferentes. Na primeira execução foi utilizado o pacote de comunicação PVM em um ambiente Linux, exclusivamente. Na segunda execução foi utilizado Java-Sockets.

2 Tolerância a Falhas em Sistemas Distribuídos

Existem alguns problemas em tornar um software tolerante a falhas. O estado da arte em tolerância a falhas em hardware diz que o hardware é muito confiável e sua confiabilidade continua aumentando com o passar do tempo. O software, por outro lado, não é confiável [JALOTE 94]. A principal causa de defeito em componentes de hardware tal como nodos e links é freqüentemente falha física.

Uma possibilidade para classificar as falhas em um sistema distribuído está baseada em como o componente falho se comporta em caso da falha. As falhas podem ser classificadas em 4 categorias: **crash**, **omissão**, **temporização** e falhas **bizantinas**, as quais estão representadas na figura 2.1 [SINGHAL 94].

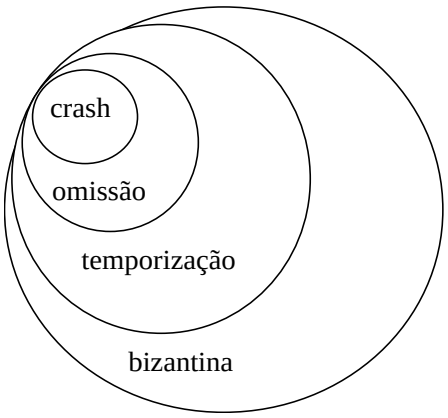

FIGURA 2.1: Classificação de Falhas [SINGHAL 94].

- **Falha de Crash**: faz com que o componente caia ou perca seu estado interno;
- **Falha de Omissão**: um componente não responde a algumas entradas;
- **Falha de Temporização**: um componente responde muito cedo ou muito tarde;
- **Falha Bizantina**: o componente comporta-se de uma forma arbitrária.

Estes quatro tipos de falhas formam uma hierarquia, com a falha de *crash* sendo a mais simples e a mais restritiva e a falha bizantina sendo menos abrangente.

2.1 Replicação de Objetos

Em uma aplicação distribuída orientada a objetos, múltiplos objetos estão cooperando para executar uma tarefa. Se um nodo falha, os objetos executando neste nodo deixam de responder. Isto pode fazer com que toda a computação distribuída venha a falhar. Os objetos devem ser implementados de forma que o processamento por eles executado continue, mesmo na ocorrência de falhas [JALOTE 94]. Uma técnica que pode ser utilizada para garantir a replicação de objetos é a primário-backup.

2.2 Técnica do Primário-Backup

Também conhecida como "Primary-Backup Approach" [ALSBERG 76]. Esta técnica utiliza um servidor primário e servidores backups do primário. Todos eles conectados por um link de comunicação [BUDHIRAJA 93] e [MULLENDER 95]. Um cliente inicialmente envia uma mensagem para o primário, conforme a indicação 1 da figura 2.2. Quando o primário recebe uma mensagem, executará as seguintes ações:

- processa a mensagem e atualiza o seu estado, se necessário;
- envia a informação sobre a atualização para o seu backup ou para os seus backups se for o caso de existir mais de um backup para o primário, conforme indicação 2 na figura 2.2;
- sem esperar por um *ack* do backup, o primário envia uma resposta para o cliente, conforme a indicação 3.
- a indicação 4 refere-se ao fato de que ocorreu uma falha no primário e cabe ao backup responder a requisição ao objeto cliente. O objeto backup sabe que o primário não está mais ativo porque não recebeu mensagens do tipo "*I'm alive*" por parte do primário (setas pontilhadas na figura 2.2), as quais são enviadas em um intervalo de tempo pré-determinado pela aplicação.

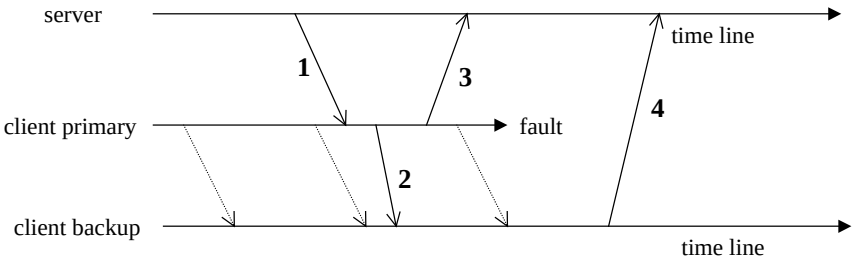

FIGURA 2.2 – Técnica do Primário-Backup.

2.3 Modelo Cliente Servidor com Sockets

O modelo cliente servidor, apresentado na Figura 2.3, pode ser utilizado com sockets. Desta forma teremos a presença de um socket no lodo do cliente, conectado a outro socket no lado do servidor. Assim, basta haver uma conexão virtual entre eles para que seja possível o tráfego das informações de do objeto cliente para o objeto servidor e vice-versa.

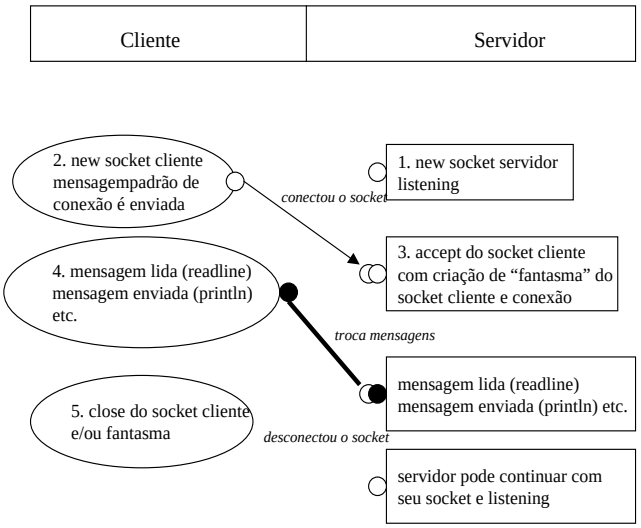

FIGURA 2.3: Modelo Cliente Servidor com Sockets.

3 Trabalhos Correlatos

Existem poucos trabalhos encontrados na literatura que implementam a replicação de objetos distribuídos utilizando-se a técnica do primário-backup. No que diz respeito a processos, aparecem as implementações de [GUERRAOUI 97], [GHOSH 97] e [ALSBERG 76]. A aplicação do primário-backup em um sistema de arquivos está presente em [BHIDE 91]. Uma configuração de hardware extra a nível de processadores, sendo um primário e outro backup, é implementada nos computadores Tandem [BARTLETT 81]. Os protocolos não-bloqueantes são o objeto do trabalho de [BUDHIRAJA 92].

O processo de replicação utilizando-se software e não um hardware especializado é defendido por [GUERRAOUI 97]. Através de um conceito intuitivo, a replicação requer técnicas sofisticadas para o sucesso da implementação. Para que seja possível atingir os objetivos da replicação, [GUERRAOUI 97] indica a necessidade de se usar um *framework* que permita a utilização de comunicação em grupo. Neste trabalho também está sendo utilizado comunicação entre grupo de processos. O ambiente do PVM – *Parallel Virtual Machine* fornece esta possibilidade, uma vez que permite a criação e a manipulação dinâmica dos grupos de processos.

Em [GHOSH 97] é utilizada a técnica primário-backup para implementar uma estratégia de escalonamento de tarefas aperiódicas em sistemas multiprocessadores *hard real-time*. O processo primário é executado em um processador e o processo backup em outro processador. Desta forma, no caso de falha do processador primário, o backup atenderá prontamente o processo que está solicitando o processamento.

Um dos protocolos que utiliza a técnica do primário-backup é o protocolo de Alsberg and Day [ALSBERG 76] . Neste protocolo, um cliente envia um pedido para o serviço e fica bloqueado esperando uma resposta. O pedido pode chegar ao primário ou ao backup. Se o pedido chegou até o primário, ele é executado. O primário atualiza o seu estado e envia o pedido até o backup e bloqueia. O backup, depois de receber a informação do primário, atualiza o seu estado, envia a resposta ao cliente e um *ack* para o primário. Ao receber o *ack*, o primário se desbloqueia e atende a outra requisição. Se o pedido chegou até o backup, então ele o enviará até o primário. O primário executará o pedido e enviará a resposta ao cliente e depois uma mensagem de atualização ao backup.

Os sistemas Tandem [BARTLETT 81] utilizam um protocolo tolerante a uma falha simples de *crash* e de *link*. Os processos são implementados aos pares, sendo que um processo nunca executa no mesmo processador que o seu par. Um dos processos é o primário e o seu par é o backup.

O protocolo HA-NFS (Highly Avaiable Network File Server) [BHIDE 91] também utiliza a técnica do primário-backup. Esta técnica usa dois servidores de disco, um é o primário e o outro é o backup. Durante a operação normal, um cliente envia um pedido ao primário, que atualiza a informação no disco e então retorna o resultado ao cliente. O primário não informa nada ao backup. A única troca de mensagens que ocorre entre os dois é do tipo "*are you alive*", que parte do backup para o primário. Quando a resposta do primário não chega, o backup assume o lugar do primário, sendo ele, agora, o servidor que atenderá às requisições dos clientes.

Também existem os protocolos não bloqueantes [BUDHIRAJA 92]. Estes protocolos possuem um interesse especial pois toleram também as falhas de omissão de recepção de mensagens.

4 Modelagem de Falhas em Aplicações com Objetos Distribuídos

O modelo tem o objetivo principal de garantir a tolerância a falhas através da replicação deste objeto. Desta forma, a falha de *crash*, no nodo onde está sendo executado um objeto, não mais fará com que todo o sistema que esteja em execução no momento da falha se perca e não consiga mais executar transações.

Para que seja possível esta implementação, será utilizada a técnica do primário-backup apresentada por Budhiraja *et al* [MULLENDER 95]. Assume-se que toda comunicação realizada entre os objetos seja através de troca de mensagens e que, na rede onde estas mensagens trafegam, não ocorra falha de *link*. Também deve ser assumido que as mensagens serão entregues de um ponto a outro da rede em um período de tempo máximo conhecido. Este modelo garante que o ambiente de execução retorne sempre a um estado global confiável e seguro, quando da presença de falha de *crash* no nodo que está processando o objeto distribuído.

Como definido na figura 4.1, a qual demonstra o modelo proposto, a função do objeto distribuído pode ser dividida em três partes distintas:

- *Criação do cliente backup*;

- *cliente primário funcionando* e
- *Restauração do cliente primário.*

O autômato finito definido na figura 4.1 descreve a primeira função do cliente como sendo a criação de seu backup.

Na primeira vez que o cliente é criado, a aplicação deverá também criar o mesmo objeto em outro nodo da rede distribuída. O cliente backup deverá estar sendo executado em outro nodo da rede. Isto é necessário para evitar a falha de ambos os clientes primário e backup no caso do nodo falhar [OLIVEIRA 98].

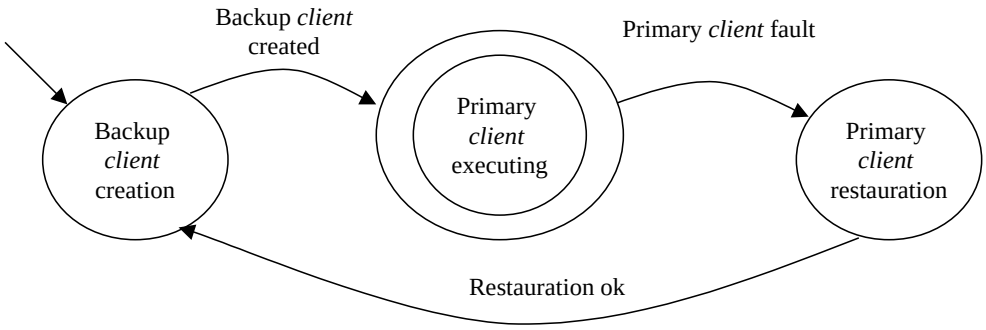

FIGURA 4.1 – Autômato Finito Determinístico Definindo o Modelo de Falha Proposto para a Aplicação Distribuída.

A segunda fase é caracterizada pela situação em que o cliente primário se encontra funcionando perfeitamente. Ele está recebendo solicitações do servidor e se comunicando com ele. Esta situação se inicia após a criação do cliente backup, conforme está demonstrado no autômato finito da figura 4.1

Como é desejado, o sistema poderá trabalhar normalmente e sem falhas. Este é o motivo pelo qual este estado, na figura 4.1, está definido como sendo o estado final do autômato finito.

A terceira fase se caracteriza pela ocorrência de falha de *crash*. Toda vez que ocorrer uma falha de *crash* no nodo que está processando o cliente primário, este estado do autômato finito será ativado. No momento da execução, esta situação será detectada pelo servidor, quando este realiza uma requisição ao cliente primário. Após decorrido um *timeout*, se não houver nenhuma resposta do cliente primário, o servidor assume que o cliente primário está em falha.

A próxima ação a ser realizada pelo servidor é fazer o mesmo pedido para o cliente backup. O cliente backup, ao receber a requisição do servidor, saberá, então, que o cliente primário não está mais respondendo. Desta forma, ele deverá criar um cliente em outro nodo da rede para que seja o seu novo cliente backup, já que agora o cliente backup atual tornou-se o novo cliente primário.

Assim, o modelo final de replicação da aplicação se encontra definido na figura 4.2. O servidor **A** faz uma requisição para o cliente primário **Cp**. O cliente primário **Cp**, por sua vez, atualiza o cliente backup **Cb** e fica aguardando um retorno desta atualização. Quando o **ACK** do cliente backup **Cb** chega, o cliente primário **Cp** envia a resposta ao servidor **A** que realizou o pedido. No modelo apresentado na figura 4.2, podem ser distinguidos três momentos nos quais uma falha poderá ocorrer [GUERRAOUI 97]. Estes momentos estão discriminados por **f1**, **f2** e **f3**.

- a falha **f1** ocorre antes de enviar a mensagem de atualização do cliente primário **Cp** para o cliente backup **Cb**;

- a falha **f2** aparece depois que a mensagem de atualização foi enviada e antes do cliente primário **Cp** receber a resposta;

- a falha **f3** ocorre após o servidor **A** receber a resposta do cliente primário **Cp**.

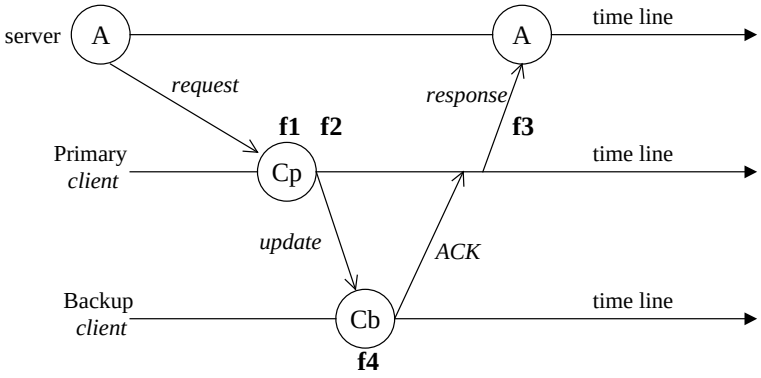

FIGURA 4.2 – Modelo de Replicação do Cliente.

Nos dois primeiros casos de falha do cliente primário, o servidor não recebe uma resposta ao seu pedido e suspeitará de uma falha. Após decorrido um *timeout*, o servidor enviará para o cliente backup o mesmo pedido feito para o cliente primário. Desta forma, os procedimentos de geração de novos clientes primário e backup serão executados.

No terceiro caso, a falha é transparente ao servidor, já que o cliente primário falhou após a resposta ter sido enviada ao servidor.

5 Implementação do Modelo

O modelo de comportamento de uma aplicação distribuída baseada em objetos comunicantes tolerantes a falhas, proposto neste trabalho, foi implementado de duas maneiras diferentes:

- a primeira implementação foi realizada em uma rede Ethernet com 5 PCs Pentium-PRO 200MHz. Foi utilizado Linux Debian 1.3, g++ da GNU e a biblioteca PVM – (Parallel Virtual Machine) versão beta 3.4.

- a segunda implementação foi feita em uma rede Ethernet composta por 5 PCs Pentium 166MHz. Foi utilizado Windows NT e Java-Sockets.

5.1 Aplicação

A aplicação realiza troca de mensagens entre os objeto cliente e os objetos distribuídos, conforme demonstrado na figura 5.1. Pode-se dividir a execução da aplicação em duas etapas distintas:

- ETAPA 1 : o servidor envia um valor para cada um dos clientes distribuídos. Fica bloqueado aguardando o retorno do resultado de uma função aplicada ao valor recebido.

- ETAPA 2: o último cliente distribuído a receber um valor do servidor será o primeiro a enviar sua resposta. O servidor enviará este valor recebido para o penúltimo cliente criado e espera o valor retornado por este para enviá-lo ao antepenúltimo cliente e assim por diante, até o primeiro cliente da aplicação.

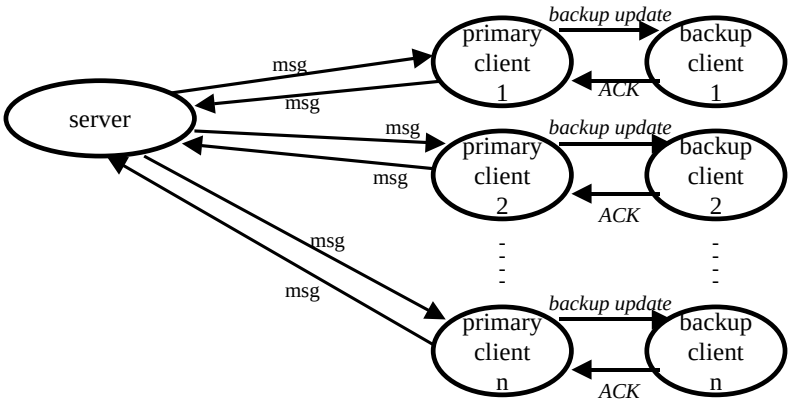

FIGURA 5.1: Troca de Mensagens entre o Servidor e os Clientes Distribuídos.

O valor retornado pelo primeiro cliente distribuído para o servidor será considerado como sendo o resultado final da aplicação.

5.2 Primeira Implementação

Com a intenção de verificar a viabilidade de implementação da replicação de objetos distribuídos utilizando-se a técnica do primário-backup, foi utilizado um ambiente composto por processos. Cada um destes processos controla a execução de um único objeto [OLIVEIRA 99].

No caso de ocorrência de alguma das falhas descritas no modelo, o método *ativa_backup ( )* será chamado. Este método se encontra apresentado na figura 5.2.

```
void ativa_backup ( ) {
        pai = dir_B;
        pvm_initsend (Pvm_Data_Default);
        pvm_pkint (&valor, 1, 1);
        pvm_send (pai, 1);
        if ((chegou = pvm_trecv (pai, 1, &tempo) > 0)
        {
           info = pvm_upkint (&valor_retorno, 1, 1);
        }
}
```

FIGURA 5.2: Método *ativa_backup ( ).*

5.3 Segunda Implementação

Esta implementação foi realizada utilizando-se Java-Sockets. A linguagem de programação Java é uma linguagem que foi criada pela Sun Microsystems, visando o desenvolvimento de aplicações na Internet e para a Internet [ARNOLD 97, ECKEL 98]. Desta forma, é apropriada para a implementação desta proposta, a qual visa atingir confiabilidade em aplicações distribuídas orientadas a objetos.

Como o Java é utilizado em aplicações cliente-servidor, tenta-se verificar o comportamento da aplicação no momento em que controla objetos (e não processos), única e exclusivamente. Esta diferença em relação à primeira implementação apresentada é fundamental, visto que, na primeira implementação, utilizam-se processos que criam e gerenciam seus objetos enquanto que, nesta implementação, não se criam processos.

A Figura 5.3 apresenta o código, em Java, utilizado para criar o objeto backup em uma máquina da rede distribuída onde a aplicação está sendo executada.

```
Backup {
                    :
                    :
                    outputS = new PrintStream (porta[i].getOutputStream( ) );
                    outputS.println (valor);
                    backup[i] = valor;
                    :
                    :
          }
```

FIGURA 5.3: Criação do Backup em Java.

5.4 Resultados Obtidos

Nas duas implementações foram realizados dois testes com a intenção de verificar o comportamento da aplicação, em relação ao tempo de execução. Este tempo é decorrente do processamento extra para se conseguir a replicação dos objetos no ambiente distribuído onde foi executada. Em cada teste, para se chegar aos resultados apresentados, foi utilizada a média aritmética dos valores de execuções da aplicação. Estas execuções aconteceram em diferentes horários [OLIVEIRA 98].

A tabela 5.1 apresenta os valores conseguidos com e sem a ocorrência de falha no cliente primário, para as implementações em PVM e em Java. Aparece também uma coluna representando a diferença de tempo entre os valores com falha e sem falha do cliente.

TABELA 5.1: Valores da Execução da Aplicação.

	PVM 05 obj.	**JAVA 05 obj.**	**PVM 10 obj.**	**JAVA 10 obj.**	**PVM 15 obj.**	**JAVA 15 obj.**	**PVM 20 obj.**	**JAVA 20 obj.**
Sem falha	1,763s	0,640s	2,483s	1,358s	3,282s	1,854s	4,025s	4,278s
Com falha	3,268s	1,317s	5,313s	2,378s	7,529s	2,323s	9,633s	4,435s
Com falha – Sem falha	1,504s	0,677s	2,829s	1,020s	4,247s	0,469s	5,608s	0,157s

O primeiro teste levou em consideração a execução da aplicação utilizando-se objetos clientes distribuídos, sem a ocorrência de falha no cliente. O objetivo deste teste é avaliar o tempo necessário para se executar a aplicação em diversas situações, cada uma contendo um número diferente de objetos comunicantes.

O gráfico 5.1 mostra os valores tempo encontrados quando se utiliza 5, 10, 15 e 20 objetos clientes distribuídos, tanto em PVM, quanto em Java. Nestes tempos se encontra computado o *overhead* causado para se conseguir a replicação do cliente.

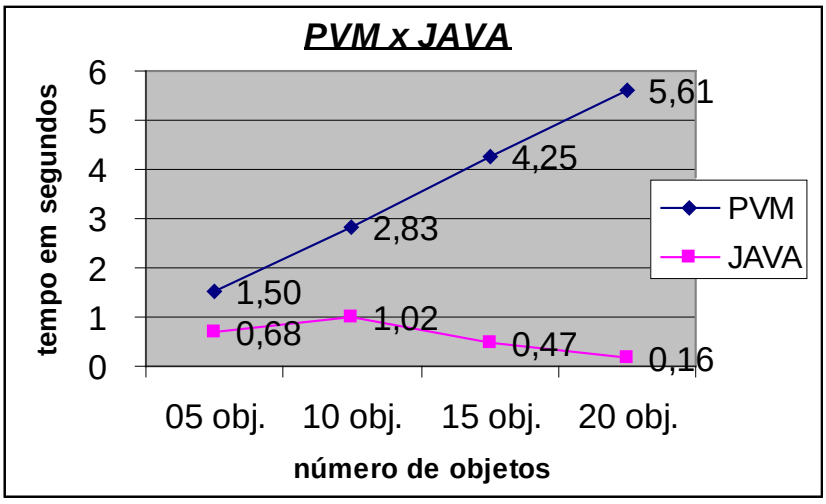

GRÁFICO 5.1: Distribuição do tempo de execução da aplicação em PVM e em Java, com a ocorrência de falhas.

6 Conclusões

Neste trabalho foi apresentado um estudo acerca do comportamento de aplicações orientadas a objetos executando em um ambiente distribuído e tolerantes a falhas. Este modelo caracterizou as falhas, descrevendo-as e inserindo-as no contexto das aplicações distribuídas. Desta forma, conseguiu-se mapear os possíveis pontos de uma transação entre objetos distribuídos que poderiam causar o encerramento forçado da aplicação.

A implementação deste modelo, realizada em PVM e em Java, garantiu que, mesmo na ocorrência de uma falha de *crash* no processador que está executando o cliente primário ou o seu backup, a aplicação continuará sua execução. Isto ficou evidenciado através dos valores apresentados na tabela 5.1 e no gráfico 5.1. Nos experimentos realizados, foi verificado que o custo extra de processamento e troca de mensagens decorrente da implementação do modelo da replicação do objeto cliente ficou em torno de 1,4s para a implementação utilizando PVM e em torno de 0,20s por objeto na implementação com Java. Desta maneira, chega-se à conclusão de que a implementação com Java é mais rápida que a implementação com PVM. Isto explica-se pelo fato de que o PVM está implementado em uma camada superior ao Java, ou seja, no Java programa-se diretamente nos sockets e o PVM usa sockets internamente para garantir a comunicação entre os processos participantes da aplicação.

O tempo alcançado, em qualquer uma das duas implementações é considerado muito pequeno, visto o ganho que se obteve ao garantir a tolerância a falhas desejada.

Avaliando os resultados obtidos verifica-se que, com a implementação do modelo de falhas apresentado neste trabalho, a disponibilidade de aplicações distribuídas, tanto em PVM quanto em Java, foi alcançada. Estas aplicações, apesar de sofrerem um acréscimo de tempo em sua execução total, conseguiram chegar ao seu final.

Pretende-se, como trabalhos futuros, utilizar uma rede composta por máquinas Sparc e por PCs. Este experimento é interessante pois poderemos verificar o comportamento de aplicações distribuídas tolerantes a falhas executando em redes heterogêneas. Outro experimento que será realizado é a utilização de outras técnicas de replicação de objetos tais como replicação ativa e mista.

Referências Bibliográficas

[ALSBERG 76] ALSBERG, P. A. and DAY, J.D. **A Principle for Resilient Sharing of Distributed Resources**. Proc. of the Second Intern. Conf. on Software Engineering. San Francisco,CA. p.562-570, 1976.

[ARNOLD 97] ARNOLD, K. **The Java Programming Language**. Massachussets, EUA. Addison-Wesley. 1997. 442p.

[BAL 89] BAL, H. E. , STEINER, J. G. and TANENBAUM, A. S. **Programming Languages for Distributed Computing Systems**. New York. ACM Computing Surveys, v.21, n.3, p.261-320, Sept. 1989.

[BARTLETT 81] BARTLETT, J. F. **A NonStop Kernel**. Proceedings of the Eighth Symposium on Operating Systems Principles. In ACM Operating Systems Review. V.15, n.5, 1981.

[BHIDE 91] BHIDE, A. ELNOZAHY, E. N. , MORGAN, S.P. **A Highly Avaiable Network File Server**. Proc. of the USENIX. P.199-205, 1991.

[BUDHIRAJA 92] BUDHIRAJA, N. and MARZULLO, K. **Tradeoffs in Implementating**

Primary-Backup Protocols. Department of Computer Science, Cornell University. Tech. Report TR 92-1307, Ithaca, Ny, 1992.

[BUDHIRAJA 93] BUDHIRAJA, N. et al. **The Primary-Backup Approach**. In: Distributed Systems. ACM Press. New York, p.199-216, 1993.

[ECKEL 98] ECKEL, B. **Thinking in Java**. N.J., EUA. PrenticeHall PTR. 1998. 1098p.

[GEIST 94] GEIST, A.et al. **PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing**. Cambridge, Massachussets: The MIT Press, 1994.

[GHOSH 97] GHOSH, S. et al. Fault-Tolerance Through Scheduling of Aperiodic Tasks in Hard Real-Time Multiprocessos Systems. **IEEE Trans. on Parallel and Distributed Systems**. New York, v. 8, n. 3, p.272-284, mar. 1997.

[GUERRAOUI 97] GUERRAOUI, R. and SCHIPER, A. **Software-Based Replication for Fault Tolerance**. In: IEEE Computer. New York, v.30, n.4, p.68-74, Apr.1997.

[JALOTE 94] JALOTE, P. **Fault Tolerance in Distributed Systems**. New Jersey: PTR Prentice Hall, Englewood Cliffs, 1994.

[MULLENDER 95] MULLENDER, S. **Distributed Systems**. Addison Wesley Publishing Company. ACM Press. New York. 1995.

[OLIVEIRA 98] OLIVEIRA JUNIOR, E. R. Replicação de Objetos em um Sistema Distribuído. In: SBAC-PAD, 10., 1998, Buzios, BRJ. **Anais...** Rio de Janeiro: COPPE / UFRJ, 1998. P.157-160.

[OLIVEIRA 99] OLIVEIRA JR, E.R. **Replicação de Objetos Distribuídos no DPC++.** Porto Alegre: PPGC/UFRGS,1999. 95p. Dissertação de Mestrado.

[SINGHAL 94] SINGHAL, M. et al. **Advanced Concepts in Operating Systems: Distributed, Database and Multiprocessor Operating Systems**. New York: McGraw-Hill, 1994.